Practica 2 TSI

José Manuel Pérez Lendinez

May 23, 2019

Contents

1 Ejercicio 1 **3**
1.1 Representación del mundo 3
1.2 Funcionalidad 4
1.3 Ejemplo utilizado 7

2 Ejercicio 2 **8**
2.1 Representación del mundo 8
2.2 Funcionalidad 8
2.3 Ejemplo utilizado 9

3 Ejercicio 3 **10**
3.1 Representación del mundo 10
3.2 Funcionalidad 11

4 Ejercicio 4 **16**
4.1 Representación del mundo 16
4.2 Funcionalidad 16

5 Ejercicio 5 **17**
5.1 Representación del mundo 17
5.2 Funcionalidad 17

6 Ejercicio 6 **17**
6.1 Representación del mundo 17
6.2 Funcionalidad 18

1 Ejercicio 1

1.1 Representación del mundo

En este caso he optado una representación muy sencilla basada en los siguientes tipos.:

1. Zonas: En este caso representan las partes de mi mapa. Y tienen la siguiente representación.

$$z1\ z2\ ...\ zn - zona$$

2. orientacion: Utilizada para saber hacia donde esta mirando el jugador y para saber que como se conectan las zonas. Su representación es :

$$norte\ sur\ este\ oeste\ -\ orientacion$$

3. Objetos y Personajes: En este caso se tendrá una linea por cada objeto o personaje en el fichero del problema que tendrá el nombre del objeto y el tipo. Como los siguientes ejemplos:

$$princesa1\ -\ princesa$$

$$princesa2\ -\ princesa$$

$$oro1\ -\ oro$$

   Para poder referenciarlos en el dominio de forma mas sencilla añadiremos los siguientes tipos:

$$princesa\ principe\ bruja\ profesor\ dicaprio\ -\ personaje$$

$$oscar\ manzana\ rosa\ algoritmo\ oro\ -\ objeto$$

4. Jugador: En este caso tendremos un tipo especifico para los jugadores como mostrare a continuación:

$$jugador1\ -\ jugador$$

$$jugador2\ -\ jugador$$

Para representar las conexiones entre zonas tengo la siguiente estructura, donde indico que esta conectado y en que punto cardinal.

$$(conectada\ norte\ z13\ z14)$$

$$(conectada\ sur\ z14\ z13)$$

Lo hago con una doble direccinalidad porque a la hora de moverme es mas sencillo buscar directamente desde la actual hasta la de destino que estar comprobando si esta guardada de una manera o otra con when. Tenemos que representar la posición que tomara un elemento en el mundo. Estos elementos pueden ser los jugadores, objetos y personajes. Para esto utilizo la siguiente estructura.

$$(posicion\_jugador\ jugador1\ z1)$$

$$(posicion\_personaje\ bruja1\ z5)$$

$$(posicion\_objeto\ oro1\ z5)$$

Lo único que nos queda por representar es la orientación del jugador que nos dirá si puede cambiar a una zona. Esto se realiza mediante el siguiente ejemplo:

$$(orientacion\_jugador\ jugador1\ norte)$$

1.2 Funcionalidad

En este caso se nos pide que:

1. Girar a izquierda: En este caso tendremos que tener en cuenta la orientación del jugador. De forma que si esta orientado al norte y gira a la izquierda su orientación cambiaría a oeste. Por tanto nos hará falta unicamente el parámetro ?player que sera de tipo jugador y mediante when y el registro de orientación del jugador se cambiara la orientación de este. Un ejemplo seria el siguiente.

   Tenemos el ?player mirando al norte. Para girarlo necesitaríamos un when que si se cumple que esta mirando al norte borre esa orientación y escriba la orientación al oeste para ?player. Este when se tendría que realizar con las 4 posibles orientaciones.

```
(:action girarDerecha
:parameters (?player - jugador)
:effect
(and
        (when (orientacion_jugador ?player norte)
                (and
                        (not(orientacion_jugador ?player norte))
                        (orientacion_jugador ?player este)
                )
        )
        (when (orientacion_jugador ?player este)
                (and
                        (not(orientacion_jugador ?player este))
                        (orientacion_jugador ?player sur)
                )
        )
        (when (orientacion_jugador ?player sur)
                (and
                        (not(orientacion_jugador ?player sur))
                        (orientacion_jugador ?player oeste)
                )
        )
        (when (orientacion_jugador ?player oeste)
```

```
                (and
                        (not(orientacion_jugador ?player oeste))
                        (orientacion_jugador ?player norte)
                )
        )
))
```

2. Girar a derecha: En este caso seria igual que el ejemplo anterior unicamente que si esta orientado por ejemplo al norte pasaría al este. La función girar a la derecha es prácticamente igual unicamente cambiando eso. Por esto no la pondré en la memoria.

3. Coger objeto: A la hora de coger un objeto hay que tener en cuenta que el jugador que lo coja y el objeto tienen que estar en la misma zona. Ademas tendremos que eliminar de la zona dicho objeto para que no pueda cogerse mas veces. Para esto como parámetros necesitaremos como hemos dicho el jugador, el objeto y la zona en la que se situan.

```
:parameters     (?player - jugador ?objeto - objeto, ?zona - zona)
```

   Con esto tendremos que comprobar que en la ?zona este el ?objeto y el ?jugador al mismo tiempo.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (posicion_objeto ?objeto ?zona)
)
```

   Con esto ya sabemos que estamos en la posición indicada tanto el jugador como el objeto. Por tanto tendremos que eliminar (posicion_objeto ?objeto ?zona) e indicar que el objeto lo tiene el ?payer. Para esto tendremos que añadir un nuevo predicado que indicara que el jugador tiene un objeto. El predicado seria el siguiente.

$$(jugador\_tiene\ ?player\ -\ jugador\ ?objeto\ -\ objeto)$$

   Con esto ya se puede realizar la opción de coger objeto.

```
:effect(and
        (not (posicion_objeto ?objeto ?zona))
        (jugador_tiene ?player ?objeto)
)
```

   En este caso como el ejercicio no indica en ningún momento que no se puedan coger mas de un objeto a la vez no he limitado esto. Aunque mas adelante esto si se especifica en otro ejercicio.

4. Dejar objeto: La mecánica de dejar objeto es muy parecida a la anterior. Tiene los mismos parámetros. Puesto que nos hace falta saber el ?player que soltara el objeto, el objeto que soltara y la zona en la que sera soltado. En este caso las precondiciones tiene que cumplir que el jugador tenga el objeto y que esta en la zona donde vamos a soltarlo.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (jugador_tiene ?player ?objeto)
)
```

A la hora de soltarlo tendremos que hacer justo lo contrario que en el caso de coger. Marcaremos que el jugador ya no tiene el objeto y que el objeto se encuentra en la zona.

```
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (posicion_objeto ?objeto ?zona)
)
```

5. Moverse entre zonas: En este caso para movernos entre zonas se tiene que cumplir que estemos mirando a la orientación correcta. Por ejemplo si estamos en z1 y esta conectada con z2 por el norte. Solo podremos pasar de z1 a z2 si el jugador esta orientado al norte. Para esto necesitaremos como parámetros la zona de destino y origen, la orientacion del jugador y el propio jugador.

```
:parameters(
        ?player - jugador
        ?cardinal - orientacion
        ?zona_actual - zona
        ?zona_destino - zona
)
```

Para poder hacer el movimiento necesitaremos estar en la ?zona_actual y orientados en la misma dirección que la ?zona_destino.

```
:precondition (and
        (orientacion_jugador ?player ?cardinal)
        (posicion_jugador ?player ?zona_actual)
        (conectada ?cardinal ?zona_actual ?zona_destino)
)
```

Si se cumplen estas precondiciones podremos movernos, para esto necesitaremos se elimine la posición actual de jugador y se cambie por la ?zona_destino.

```
:effect(and
        (not (posicion_jugador ?player ?zona_actual))
        (posicion_jugador ?player ?zona_destino)

)
```

1.3 Ejemplo utilizado

El mapa que usare en todos los ejercicios es este.

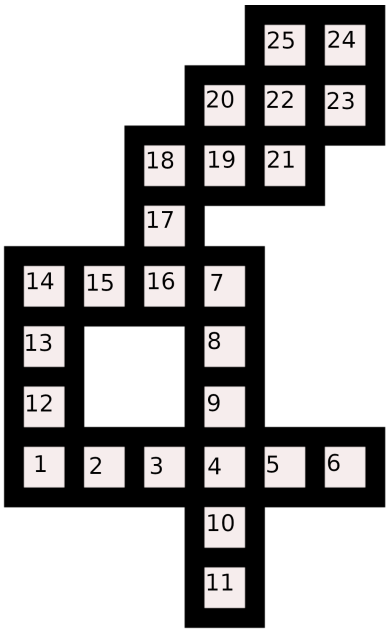


Figure 1

En caso hemos colocado un personaje de cada tipo, un jugador y un objeto de cada tipo.

Tipo	Nombre	Posición
oro	oro1	25
oscar	oscar1	18
rosa	rosa1	9
algoritmo	algoritmo1	12
manzana	manzana1	11
princesa	princesa1	7
principe	principe1	10
profesor	profesor1	20
bruja	bruja1	5
dicaprio	dicaprio1	6
jugador	jugador	1

Como goal puse que entregaran a un objeto a cada personaje. Sin importar el objeto que sea. Fue ejecutado sin -O.

2 Ejercicio 2

2.1 Representación del mundo

El ejercicio nos pide añadir costes a los caminos entre zonas. De forma que moverse de z1 a z2 tenga un coste x. Para esto necesitaremos añadir nuevos predicados que nos informen del coste entre las zonas. Para esto he optado por añadir un nuevo predicado de tipo función donde se almacenara el coste del camino entre dos zonas. La estructura es la siguiente:

$$(= \ (coste \ z4 \ z3) \ 1)(= \ (coste \ z3 \ z4) \ 1)$$

Esto identifica las dos zonas y el coste es el 3 parámetro. En este caso pasar de la zona 4 a la zona 3 tendría un coste de 1. Uso la doble direccionalidad también para ahorrarme comprobaciones al mirar los costes.

También tendremos que añadir un nuevo predicado de tipo función para llevar la cuenta del coste total.

$$(= \ (coste\_total) \ 0)$$

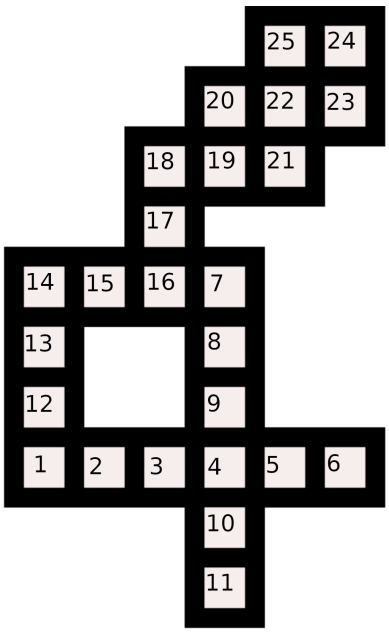
2.2 Funcionalidad

En este caso los cambios que se realizan son muy pequeños. Tenemos que ir sumando el coste del arco a la hora de movernos con la función para moverse. Por tanto solo tendríamos que añadir a esta función la linea que nos incremente el coste total. La funcino quedaria de la siguiente manera.

```
:effect(and
        (not (posicion_jugador ?player ?zona_actual))
        (posicion_jugador ?player ?zona_destino)
        (increase (coste_total) (coste ?zona_actual ?zona_destino))

)
```

2.3 Ejemplo utilizado



En caso hemos colocado un personaje de cada tipo, un jugador y un objeto de cada tipo.

Tipo	Nombre	Posición
oro	oro1	25
oscar	oscar1	18
rosa	rosa1	9
algoritmo	algoritmo1	12
manzana	manzana1	11
princesa	princesa1	7
principe	principe1	10
profesor	profesor1	20
bruja	bruja1	5
dicaprio	dicaprio1	6
jugador	jugador	1

En este caso obligue mediante el goal que a cierto personaje le diera cierto objeto. Hice esto porque sino tardo mucho en ejecutar al tener muchas opciones y optimizar con -O

```
(:goal (AND
        (personaje_tiene princesa1 rosa1)
        (personaje_tiene principe1 oro1)
        (personaje_tiene profesor1 algoritmo1)
        (personaje_tiene bruja1 manzana1)
        (personaje_tiene dicaprio1 oscar1)
))
(:metric minimize (coste_total))
```

Los pesos de los arcos son los siguientes:

Zona	Zona	coste
z1	z2	1
z2	z3	3
z3	z4	1
z4	z5	1
z4	z9	1
z4	z10	3
z5	z6	1
z7	z8	2
z8	z9	1
z10	z11	2
z12	z13	1
z13	z14	1
z14	z15	2
z15	z16	1
z16	z17	4
z16	z7	3
z17	z18	1
z18	z19	2
z19	z20	1
z19	z21	2
z20	z22	1
z21	z22	1
z22	z23	3
z22	z25	2
z23	z24	1
z24	z25	2

3 Ejercicio 3

3.1 Representación del mundo

Se añade a la zona una nueva características que representara el tipo de terreno. Para representar el tipo de terreno de una zona se necesitara un nuevo predicado.

El predicado contendrá la zona y el tipo de terreno.

$$(tipo\_terrenoz5piedra)$$

Al añadirse nuevos terrenos tendremos que crear un nuevo tipo que los contendrá para poder trabajar con ellos. El tipos es el siguiente:

$$piedra\ agua\ bosque\ arena\ precipicio\ -\ suelos$$

Se añaden dos nuevos tipos de objetos(bikini y zapatillas). En este caso los objetos se registran como en los casos anterior.

$$bikini1\ -\ bikini$$

$$zapatillas1\ -\ zapatillas$$

En este caso tendremos que modificar también el tipo de objetos explicado anteriormente para incluir los nuevos. Quedando de la siguiente manera.

$$oscar\ manzana\ rosa\ algoritmo\ oro\ bikini\ zapatilla\ -\ objeto$$

También se añade una mochila al usuario y se limita el numero de objetos que se pueden coger a 1 en las manos y otro en la mochila. Según el enunciado especifica que para meter un objeto en la mochila tiene que tenerlo antes en las manos. A la hora de sacarlo en la mochila no decía nada al respecto. Por lo que yo he añadido un sacar de la mochila que deja el objeto directamente en el suelo. Se limita a 1 el numero de objetos que podemos tener en la mano.

Para esto he añadido dos nuevos predicados uno para indicar que tiene un objeto en la mochila y otro para indicar que tiene un objeto cogido.

$$(jugador\_sin\_objeto\ jugador1)$$

$$(mochila\_vacia\ jugador1)$$

El primero indica que no tiene objetos en las manos y el segundo que no tiene objetos en la mochila.

Con esto ya tenemos toda la representación del problema realizada.

3.2 Funcionalidad

Empecemos por la parte de la mochila.

En este caso la mochila afecta a funciones anteriores. Vamos a analizarlas una a una. Las funciones anteriores que tendremos que modificar son las siguientes.

1. Coger objeto: Anteriormente no teníamos limitado el numero de objetos que podía coger el jugador. Por esto he tenido que añadir a las precondiciones que para coger un objeto el jugador se tiene que tener el predicado jugador_sin_objetos activo.

```
:precondition(and
        (jugador_sin_objeto ?player)
        (posicion_jugador ?player ?zona)
        (posicion_objeto ?objeto ?zona)
)
```

Ademas a la hora de coger el objeto en el apartado effect tendremos que eliminar ese predicado para indicar que ya tenemos un objeto en las manos.

```
:effect(and
        (not (posicion_objeto ?objeto ?zona))
        (not (jugador_sin_objeto ?player))
        (jugador_tiene ?player ?objeto)

)
```

Con estas pequeñas modificaciones la acción coger objeto ya esta preparada.

2. Dejar objeto: La modificación en este caso es muy parecida a lo explicado en coger objeto. En este caso se tendrá que ver que tenemos un objeto en la mano. Esta parte no cambia el cambio viene en el apartado de effect donde añadiremos la regla (jugador_sin_objeto ?player) que indicara que el jugador a soltado el objeto.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (jugador_tiene ?player ?objeto)

)
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (posicion_objeto ?objeto ?zona)
        (jugador_sin_objeto ?player)
)
```

3. Entregar objeto: En este caso tendremos que tener en cuenta en el apartado de effects que el jugador solo puede entregar objetos que tiene en la mano y por tanto una vez entregado se quedara sin el objeto y con las manos libres. Para esto solo modificaremos la parte de :effect quedando las precondition como las teníamos anteriormente. En la parte de precondition tendremos que añadir el predicado jugador_sin_objetos para indicar que el jugador tiene las manos libres.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (posicion_personaje ?personaje ?zona)
        (jugador_tiene ?player ?objeto)
)
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (jugador_sin_objeto ?player)
        (personaje_tiene ?personaje ?objeto)
        (personaje_tiene_objeto ?personaje)
)
```

También tendremos que añadir dos nuevas funciones para controlas la mochila. Estas funciones son:

1. Meter mochila: Los necesarios serán le jugador y el objeto unicamente puesto que solo se añadirá el objeto a la mochila.

```
:parameters (?player - jugador ?objeto - objeto)
```

   Las precondiciones que tendremos que tener en cuenta es que la mochila este vaciá y que el jugador tenga el objeto en la mano. Esto podemos conseguirlos usando los predicados mochila_vacia y jugador_tiene.

```
:precondition(and
        (mochila_vacia ?player)
        (jugador_tiene ?player ?objeto)

)
```

   Una vez se cumplan las precondiciones tendremos que añadir que el jugador no tiene objeto en las manos y que la mochila tiene un nuevo objeto y no esta vaciá.

```
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (not (mochila_vacia ?player))
        (mochila_tiene ?player ?objeto)
        (jugador_sin_objeto ?player)

)
```

2. Sacar de mochila: En este caso como explique anteriormente vaciara la mochila en el suelo. Para ello necesitaremos como parámetros aparte del objeto y el jugador, la zona en la que nos encontramos.

```
:parameters (?player - jugador ?objeto - objeto ?zona - zona)
```

   Las precondiciones en este caso son que el jugador se encuentre en dicha zona y que tenga un objeto en la mochila.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (mochila_tiene ?player ?objeto)

)
```

   Con esto ya pasamos a la parte de effect donde tendremos que indicar que la mochila esta vacía y el objeto a quedado en el suelo.

```
:effect(and
        (mochila_vacia ?player)
        (not (mochila_tiene ?player ?objeto))
        (posicion_objeto ?objeto ?zona)

)
```

Con estos cambios ya tenemos metido todo lo necesarios para que el jugador pueda tener un objeto en la mochila y otro en la mano.

La segunda parte de este ejercicio sera las restricciones dadas por los tipos de terreno de cada zona. Esta restricciones son las siguiente.

1. Bosque: Para pasar por el bosque tendremos que tener en cuenta que el jugador necesita tener en las manos o en la mochila el objeto zapatilla. Para esto tendremos que crear una nueva acción para moverse atrevas del bosque.

   Los parámetros necesarios para saber esto son el jugador, su orientación , la zona actual y de destino, el tipo de terreno de la zona de destino y los objetos de zapatillas para ver si tenemos alguno.

```
:parameters(
         ?player - jugador
         ?cardinal - orientacion ?zona_actual - zona ?zona_destino -
              zona
         ?terreno - suelos
         ?objeto - zapatilla
)
```

   Las precondiciones utilizadas en este caso son muy parecidas a las del ejercicio1 añadiendo un predicado para que el jugador tenga las zapatillas en la mano o en la mochila.

```
:precondition (and
(orientacion_jugador ?player ?cardinal)
        (posicion_jugador ?player ?zona_actual)
        (conectada ?cardinal ?zona_actual ?zona_destino)
        (tipo_terreno ?zona_destino ?terreno)
        (or (jugador_tiene ?player ?objeto) (mochila_tiene ?player ?
             objeto))

)
```

   Si se cumplen las precondiciones tendremos que mirar que el terreno de la zona de destino sea bosque con un when y si se cumple el jugador podrá avanzar.

```
:effect(and
        (when (tipo_terreno ?zona_destino bosque)
                (and
                        (not (posicion_jugador ?player ?zona_actual))
                        (posicion_jugador ?player ?zona_destino)
                        (increase (coste_total) (coste ?zona_actual ?
                             zona_destino))
                )
        )
)
```

2. Agua: En este caso sera prácticamente igual que en el caso anterior cambiando que el objeto sea de tipo bikini y que la zona sea de agua. Como se muestra en el siguiente código.

```
(:action moverseOrientadoAgua
        :parameters(
                ?player - jugador
                ?cardinal - orientacion
                ?zona_actual - zona
                ?zona_destino - zona
                ?terreno - suelos
                ?objeto - bikini
        )
        :precondition (and
                (orientacion_jugador ?player ?cardinal)
                (posicion_jugador ?player ?zona_actual)
                (conectada ?cardinal ?zona_actual ?zona_destino)
                (tipo_terreno ?zona_destino ?terreno)
                (or (jugador_tiene ?player ?objeto) (mochila_tiene ?
                    player ?objeto))

        )
        :effect(and

                (when (tipo_terreno ?zona_destino agua) (and

                        (not (posicion_jugador ?player ?zona_actual))
                        (posicion_jugador ?player ?zona_destino)
                        (increase (coste_total) (coste ?zona_actual ?
                            zona_destino))
                ))
        )
)
```

3. Arena y piedra: Para pasar por este terreno no se necesita ningún tipo de objeto. Por tanto las precondiciones y las parámetros son los mismo que en el ejercicio 1 añadiendo unicamente el predicado (tipo_terreno ?zona_destino ?terreno) para saber el terreno que tendremos y en los parámetros añadimos ?terreno - suelo para obtener el tipo de suelo.

   Depures la parte effects es exactamente igual que en el movimiento en bosque y agua, simplemente añadiendo dos when en vez de uno y que uno de ellos se active si el terreno es arena y otro si es roca.

4. Precipicio: El jugador no puede pasar por precipicio. Esto se cumple a no tener añadido en ninguno de las tres funciones de movimiento un when que tenga el cuenta el tipo de suelo Precipicio. Por tanto no hay que añadir nada nuevo para esta parte.

4 Ejercicio 4

4.1 Representación del mundo

Para representar los valores de la tabla dada he añadido un predicado de tipo funcion con la siguiente estructura.

$$(= \ (valor\_objeto \ rosa1 \ princesa1) \ 10)$$

Donde princesa1 es un personaje y rosa1 es un objeto. seguido de la puntuación que se obtendría al entregarle ese objeto.

También añado un nuevo predicado de tipo función con los puntos que tenemos que conseguir y otro para almacenar los puntos actuales.

$$(= \ (puntos\_minimos) \ 50)$$

$$(= \ (puntos\_totales \ jugador1) \ 0)$$

4.2 Funcionalidad

En este caso la funcionalidad a cambiar se encuentra en la acción que entrega los objetos a el jugador. En este caso solo hay que añadir una nueva linea en la parte de efectos para que incremente los puntos totales respecto al objeto entregado. El effect quedaría de la siguiente manera.

```
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (jugador_sin_objeto ?player)
        (personaje_tiene ?personaje ?objeto)
        (personaje_tiene_objeto ?personaje)
        (increase (puntos_totales ?player) (valor_objeto ?objeto ?personaje))
)
```

En este caso el goal tendría que cumplir que la puntuación total sea igual que la puntuación pasada como mínima.

```
(:goal (AND(= (puntos_totales jugador1) (puntos_minimos))))
```

5 Ejercicio 5

5.1 Representación del mundo

En este caso se añade un nuevo predicado de tipo función que tendrá el numero de objetos que acepta un personaje. El predicado tendría la siguiente estructura.

$$(= (bolsilloprincesa1)5)$$

Princesa1 representaría al personaje y 5 seria el numero máximo de objetos que podría tener. No necesito tener ningún predicado mas porque en la parte de funcionalidad voy restando a este.

5.2 Funcionalidad

En este caso solo se modificara la acción entregar objeto. En esta acción se modificara la parte de precondition y la parte de effect. En precondition tendremos que comprobar que el numero del predicado (bolsillo ?personaje) sea mayor que 0. Esto nos indicara que aun puede tener mas objetos. Si se cumple esto entregaremos el objeto al personaje y decrementaremos el bolsillo en 1. Este decremento se hará en la parte de effects.

```
:precondition(and
        (posicion_jugador ?player ?zona)
        (posicion_personaje ?personaje ?zona)
        (jugador_tiene ?player ?objeto)
        (> (bolsillo ?personaje) 0)
)
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (jugador_sin_objeto ?player)
        (personaje_tiene ?personaje ?objeto)
        (personaje_tiene_objeto ?personaje)
        (increase (puntos_totales ?player) (valor_objeto ?objeto ?personaje))
        (decrease (bolsillo ?personaje) 1)

)
```

6 Ejercicio 6

6.1 Representación del mundo

Se introducen 3 nuevo tipos de predicados. El primero seria el que marca los putos que hay que conseguir entre varios personajes.

$$(= \ (puntos\_minimos) \ 10)$$

El valor numérico indicaría el numero de puntos mínimos a conseguir. Después tendremos que tener otro que usaremos como contador de puntos alcanzados. En este todos los personajes añadirán los puntos que van consiguiendo al entregar un objeto

$$(= \ (puntos\_conjuntos) \ 0)$$

También necesitaremos otro que actué como contador individual para cada personaje. De forma que cada vez que entregue un objeto se cuente en este. Este predicado yo ya lo tenia puesto que anteriormente ya que hice el problema pensando en la posibilidad de que se tengan mas de un jugador. Es el siguiente:

$$(= \ (puntos\_totales \ jugador2) \ 0)$$

El ultimo que he introducido ha sido el que se encarga de tener el mínimo de puntos para cada personaje.

$$(= \ (puntos\_jugador\_objetivo \ jugador2) \ 5)$$

El goal en este caso tiene que cumplir varios requisitos. Que todos los jugadores superen su limite de puntos y que en conjunten superen también el mínimo conjunto. Seria el siguiente:

```
(:goal (AND
        (> (puntos_conjuntos) (puntos_minimos))
        (> (puntos_totales jugador1) (puntos_jugador_objetivo jugador1))
        (> (puntos_totales jugador2) (puntos_jugador_objetivo jugador2))
))
```

6.2 Funcionalidad

En este caso como en el ejercicio anterior lo único que he tenido que modificar es la parte de effects de la acction que entrega objetos a los personajes. El único cambio en mi caso es añadir un incremento que aumente el contador compartido.

```
:effect(and
        (not (jugador_tiene ?player ?objeto))
        (jugador_sin_objeto ?player)
        (personaje_tiene ?personaje ?objeto)
        (personaje_tiene_objeto ?personaje)
        (increase (puntos_totales ?player) (valor_objeto ?objeto ?personaje))
        (increase (puntos_conjuntos) (valor_objeto ?objeto ?personaje))
        (decrease (bolsillo ?personaje) 1)

)
```